

Semánticas de lenguajes

Especificación de un lenguaje declarativo con maude

Sergio García Sánchez

UCM

10 de diciembre de 2021

- 1 Introducción
- 2 Sintaxis
- 3 Semántica
- 4 Ejecución
- 5 Ejemplos
- 6 Conclusiones

- Definir la sintaxis de un lenguaje
- Definir la semántica de un lenguaje mediante reglas de reescritura
- Uso de la librería de maude para python

```
sorts Pair Memory .
subsort Pair < Memory .

op [_,_] : Qid Int -> Pair [ctor] .
op none : -> Memory [ctor] .
op -- : Memory Memory -> Memory [ctor assoc comm id: none]

op _ [_\_] : Memory Qid Int -> Memory .
eq ( M [ Q , V ] ) [ Q \ V' ] = [ Q , V' ] M .
eq M [ Q \ V ] = [ Q , V ] M [owise] .

op find : Memory Qid -> Int .
eq find (M [Q, V] M', Q) = V .
eq find (M, Q) = 0 [owise] .
```

```
sorts Type Index .  
subsorts Qid Int Index < Type .  
  
op _<_> : Type Type -> Index [ctor] .  
  
op load : Memory Type -> Type .  
eq load(M, Q) = find(M, Q) .  
eq load(M, I) = I .  
eq load(M, In) = find(M, toQid(In, M)) .
```

Expresiones

```
sort Expression .
subsort Type < Expression .

op _sum_ : Expression Expression -> Expression [ctor assoc comm] .
op _less_ : Expression Expression -> Expression [ctor] .
op _mult_ : Expression Expression -> Expression [ctor assoc comm] .

op eval : Expression Memory -> Type .
eq eval (T sum T', M) = load(M,T) + load(M,T') .
eq eval (T less T', M) = load(M,T) - load(M,T') .
eq eval (T mult T', M) = load(M,T) * load(M,T') .
eq eval (T, M) = load(M, T) [owise] .
```

```
sorts Condition MultipleCondition .  
subsort MultipleCondition < Condition .
```

```
op _eq_ : Expression Expression -> Condition [ctor] .  
op _neq_ : Expression Expression -> Condition [ctor] .  
op _gt_ : Expression Expression -> Condition [ctor] .  
op _gte_ : Expression Expression -> Condition [ctor] .  
op _lt_ : Expression Expression -> Condition [ctor] .  
op _lte_ : Expression Expression -> Condition [ctor] .
```

```
op _&&_ : Condition Condition -> MultipleCondition [ctor assoc comm] .  
op _||_ : Condition Condition -> MultipleCondition [ctor assoc comm] .
```

```
op eval : Condition Memory -> Bool .
eq eval(E eq E', M) = load(M, eval(E, M)) == load(M, eval(E', M)) .
eq eval(E neq E', M) = load(M, eval(E, M)) /= load(M, eval(E', M)) .
eq eval(E gt E', M) = load(M, eval(E, M)) > load(M, eval(E', M)) .
eq eval(E gte E', M) = load(M, eval(E, M)) >= load(M, eval(E', M)) .
eq eval(E lt E', M) = load(M, eval(E, M)) < load(M, eval(E', M)) .
eq eval(E lte E', M) = load(M, eval(E, M)) <= load(M, eval(E', M)) .

eq eval(C && C', M) = eval(C, M) and eval(C', M) .
eq eval(C || C', M) = eval(C, M) or eval(C', M) .
```



```
sort Assig .
```

```
op _=_ : Type Expression -> Assig [ctor] .
```

```
op [ _ < _ > < _ > ] = _ : Qid Type Type Expression -> Assig [ctor]
```

```
op _=[_] : Type Array -> Assig [ctor] .
```

```
op _= size(_) : Type Qid -> Assig [ctor] .
```

```
op assigInMemory : Assig Memory -> Memory .
```

```
eq assigInMemory(Q = E, M) = M [Q \ load(M, eval(E, M))] .
```

```
eq assigInMemory(In = E, M) = M [toQid(In, M) \ load(M, eval(E, M))]
```

```
sort Program .  
op end : -> Program [ctor] .  
op //_ : Program -> Program [ctor] .  
op (_); : Assig -> Program [ctor] .  
op print(_); : Type -> Program [ctor] .  
op println(_); : Type -> Program [ctor] .  
op print(_); : String -> Program [ctor] .  
op println(_); : String -> Program [ctor] .  
op scanf(_,_); : Type String -> Program [ctor] .  
op (_=futureRead); : Type -> Program [ctor] .
```

```
op if(_) {_} : Condition Program -> Program [ctor] .
op if(_) {_} else {_} : Condition Program Program -> Program [ctor]
op while(_) {_} : Condition Program -> Program [ctor] .
op do {_} while(_); : Program Condition -> Program [ctor] .
op for((_);(_);(_);) {_} : Assig Condition Assig Program -> Program
op forWithoutInitial((_);(_);) {_} : Condition Assig Program -> Program
op __ : Program Program -> Program [ctor assoc id: end] .
```

```
sort System .  
subsort System < Attribute .  
op {_|_} : Program Memory -> System [ctor] .  
  
op System : -> Cid .  
op system : -> Oid .
```

```
rl [coment] :  
{ // PBody P | M}  
=>  
{ P | M} .
```

```
rl [printType] :  
  < system : System | { print(T); P | M } >  
  =>  
  < system : System | { P | M } >  
  write(stdout, system, string(load(M, T), 10)) .
```

```
rl [printlnType] :  
  < system : System | { println(T); P | M } >  
  =>  
  < system : System | { P | M } >  
  write(stdout, string(load(M, T) + "\n", 10), system) .
```

```
rl [printString] :  
  < system : System | { print(S); P | M } >  
  =>  
  write(stoud, S, system) < system : System | { P | M } > .
```

```
rl [startScanf] :  
  < system : System | { scanf(T, S); P | M } >  
  =>  
  getLine (stdin, system, S)  
  < system : System | { (T =futureRead); P | M } > .  
  
crl [endScanf] :  
  < system : System | { (T =futureRead); P | M } >  
  gotLine (system, 0, S)  
  =>  
  < system : System | { P | assigInMemory(T = T', M) } >  
  if S /= ""  
  /\ Length := length(S)  
  /\ Length' := sd(Length, 1)  
  /\ T' := rat(substr(S, 0, Length'), 10) .
```

```
rl [assig] :  
  { (T = E); P | M}  
=>  
  { P | assignInMemory(T = E, M) } .
```

```
rl [assigSize] :  
  { (T = size(Q)); P | M}  
=>  
  { P | assignInMemory(T = size(M, Q, 0), M) } .
```

```
rl [assigArray] :  
  { (T = [Array]); P | M}  
=>  
  { P | assignInMemory(T = [Array], M, 0) } .
```



```
cr1 [ifTrue] :  
  { if(C){PBody} P | M }  
=>  
  { PBody P | M }  
  if eval(C, M) .
```

```
cr1 [ifFalse] :  
  { if(C){PBody} P | M }  
=>  
  { P | M }  
  if not eval(C, M) .
```

```
cr1 [ifElseTrue] :  
  { if(C){PBody}else{PBodyElse} P | M }  
=>  
  { PBody P | M }  
  if eval(C, M) .
```

```
cr1 [ifElseFalse] :  
  { if(C){PBody}else{PBodyElse} P | M }  
=>  
  { PBodyElse P | M }  
  if not eval(C, M) .
```

While

```
cr1 [whileTrue] :  
  { while(C){PBody} P | M }  
=>  
  { PBody while(C){PBody} P | M }  
if eval(C, M) .
```

```
cr1 [whileFalse] :  
  { while(C){PBody} P | M }  
=>  
  { P | M }  
if not eval(C, M) .
```

```
rl [doWhile] :  
  { do {PBody} while(C); P | M }  
=>  
  {PBody while(C){PBody} P | M} .
```

```

rl [initalFor] :
  { for( (A); (C); (A'); ){PBody} P | M }
  =>
  { (A); forWithoutInitial((C); (A'); ){PBody} P | M } .

cr1 [forTrue] :
  { forWithoutInitial((C); (A'); ){PBody} P | M }
  =>
  { PBody (A'); forWithoutInitial((C); (A'); ){PBody} P | M }
  if eval(C, M) .

cr1 [forFalse] :
  { forWithoutInitial((C); (A'); ){PBody} P | M }
  =>
  { P | M }
  if not eval(C, M) .

```

```
import maude

def main(filename):
    programFile = open(filename, "r").read()

    system = " < system : System [{ ${program} | none } > "
    generatedSystem = system.replace("${program}", programFile)

    maude.init()
    maude.load("src/loads.maude")
    maude.getModule('SEMANTICS')
        .parseTerm(generatedSystem)
        .erewrite()
```

Ejemplos

```
scanf('n, "Introduce la longitud del vector: ");
for(('i = 0); ('i lt 'n); ('i = 'i sum 1);){
    scanf('v < 'i > , "Introduce un numero: ");
}
print("Desordenado: ");
for(('i = 0); ('i lt 'n); ('i = 'i sum 1);) {
    print('v < 'i >);
    print(" ");
}
for(('i = 0); ('i lt 'n); ('i = 'i sum 1);) {
    for(('j = 1); ('j lt ('n less 'i)); ('j = 'j sum 1);) {
        ('ant = 'j less 1);
        if('v < 'ant > gt 'v < 'j >) {
            ('tmp = 'v < 'ant >);
            ('v < 'ant > = 'v < 'j > );
            ('v < 'j > = 'tmp);
        }
    }
}
print("Ordenado: ");
for(('i = 0); ('i lt 'n); ('i = 'i sum 1);) {
    print('v < 'i >);
    print(" ");
}
```

Bubble Sort

```
!master ~/workspace/language> ./language.sh
```

```
Introduce la longitud del vector: 5
```

```
Introduce un numero: 5
```

```
Introduce un numero: 4
```

```
Introduce un numero: 3
```

```
Introduce un numero: 2
```

```
Introduce un numero: 1
```

```
Desordenado: 5 4 3 2 1
```

```
Ordenado: 1 2 3 4 5
```

Conclusiones